МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет физики, математики, информатики
Кафедра программного обеспечения и администрирования информационных систем

КУРСОВОЙ ПРОЕКТ
по дисциплине
структуры и алгоритмы компьютерной обработки данных
на тему: ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ДВОИЧНОГО ДЕРЕВА ПОИСКА

Обучающегося 2 курса
очной формы обучения
направления подготовки
02.03.03 Математическое обеспечение
и администрирование
информационных систем
Направленность (профиль)
Проектирование информационных
систем и баз данных
Козявина Максима Сергеевича

Руководитель:
профессор кафедры ПОиАИС
Кудинов Виталий Алексеевич

Допустить к защите:
_____ / _____
«_____» _____ 20_____г.

Курск, 2023

СОДЕРЖАНИЕ

1 Задание

1.1 Формулировка условия задачи

Разработать программную реализацию двоичного дерева поиска, содержащую статический массив указателей на строки, 2 указателя (ссылки) на потомков и счетчик вершин в поддереве.

Реализовать полный набор операций (добавление, включение и извлечение по логическому номеру, включение с сохранением порядка, загрузка и сохранение строк в текстовом файле, балансировка – выравнивание размерностей структур данных нижнего уровня).

Программа должна быть реализована с использованием указателей (ссылок) и динамической памяти. Также необходимо провести тестирование программы на различных тестовых данных, оценить ее эффективность и оптимизировать при необходимости.

1.2 Краткие теоретические сведения об объекте исследования

Двоичное дерево поиска – это структура данных, которая представляет собой дерево, в котором каждый узел имеет не более двух потомков. Каждый узел содержит ключ и связанные с ним данные. Узлы с ключами меньше, чем узел-родитель и находятся в левом поддереве, а узлы с большими ключами – в правом поддереве.

Основные операции над двоичным деревом поиска:

1. Добавление нового элемента в дерево:

– Если дерево пустое, то новый элемент становится корнем дерева;

– Если элемент меньше корня, то он добавляется в левое поддерево, если больше – в правое;

– Рекурсивно повторяем предыдущий шаг до тех пор, пока не найдется подходящее место для нового элемента.

2. Удаление элемента из дерева:

– Если удаляемый элемент – листовой узел, то он просто удаляется;

– Если удаляемый элемент имеет одного потомка, то этот потомок заменяет его в дереве;

– Если удаляемый элемент имеет двух потомков, то его место занимает наименьший элемент из правого поддерева, а затем он удаляется из правого поддерева.

3. Поиск элемента в дереве:

– Сравниваем ключ искомого элемента с ключом корня;

– Если ключ меньше, чем ключ корня, то ищем в левом поддереве;

– Если ключ больше, чем ключ корня, то ищем в правом поддереве;

– Если ключ равен ключу корня, то элемент найден.

4. Обход дерева в прямом, обратном и симметричном порядках:

– Прямой порядок: корень, левое поддерево, правое поддерево;

– Обратный порядок: правое поддерево, левое поддерево, корень;

– Симметричный порядок: левое поддерево, корень, правое поддерево.

5. Вывод дерева на экран в виде графической структуры:

– Используем обратный порядок обхода дерева;

– Для каждого узла выводим его ключ и данные;

– Для каждого узла выводим его потомков с отступом.

Для реализации двоичного дерева поиска используется динамическая память и указатели. Каждый узел дерева представляется структурой, содержащей ключ, данные и указатели (ссылки) на левого и правого потомков. Для добавления нового элемента выделяется память под новый узел и производится его инициализация. Для удаления элемента используется рекурсивный подход. Для поиска элемента также используется рекурсивный подход. Для обхода дерева используется рекурсивный алгоритм, который вызывает функцию обхода для левого потомка, затем для правого потомка и выводит данные текущего узла. Для вывода дерева на экран используется обратный порядок обхода и отступы для каждого узла.

Тестирование программы проводится на различных тестовых данных, включая случайные, отсортированные и обратно отсортированные массивы. Оценка эффективности программы осуществляется с помощью анализа времени выполнения операций и использования памяти. При необходимости производится оптимизация программы для улучшения ее производительности и эффективности.

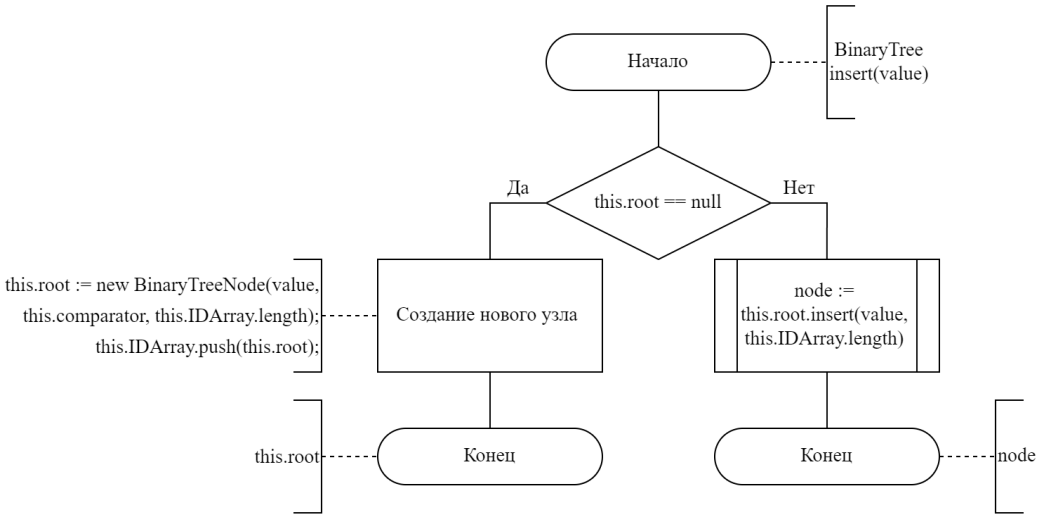

Рисунок 1 – Алгоритм метода вставки узла (класс BinaryTree)

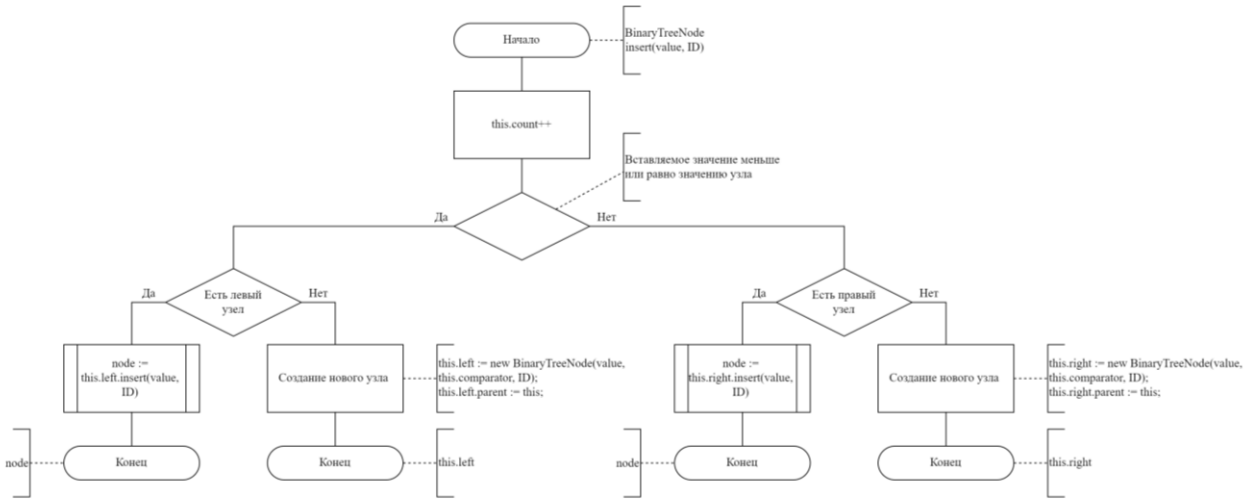

Рисунок 2 – Алгоритм метода вставки узла (класс BinaryTreeNode)

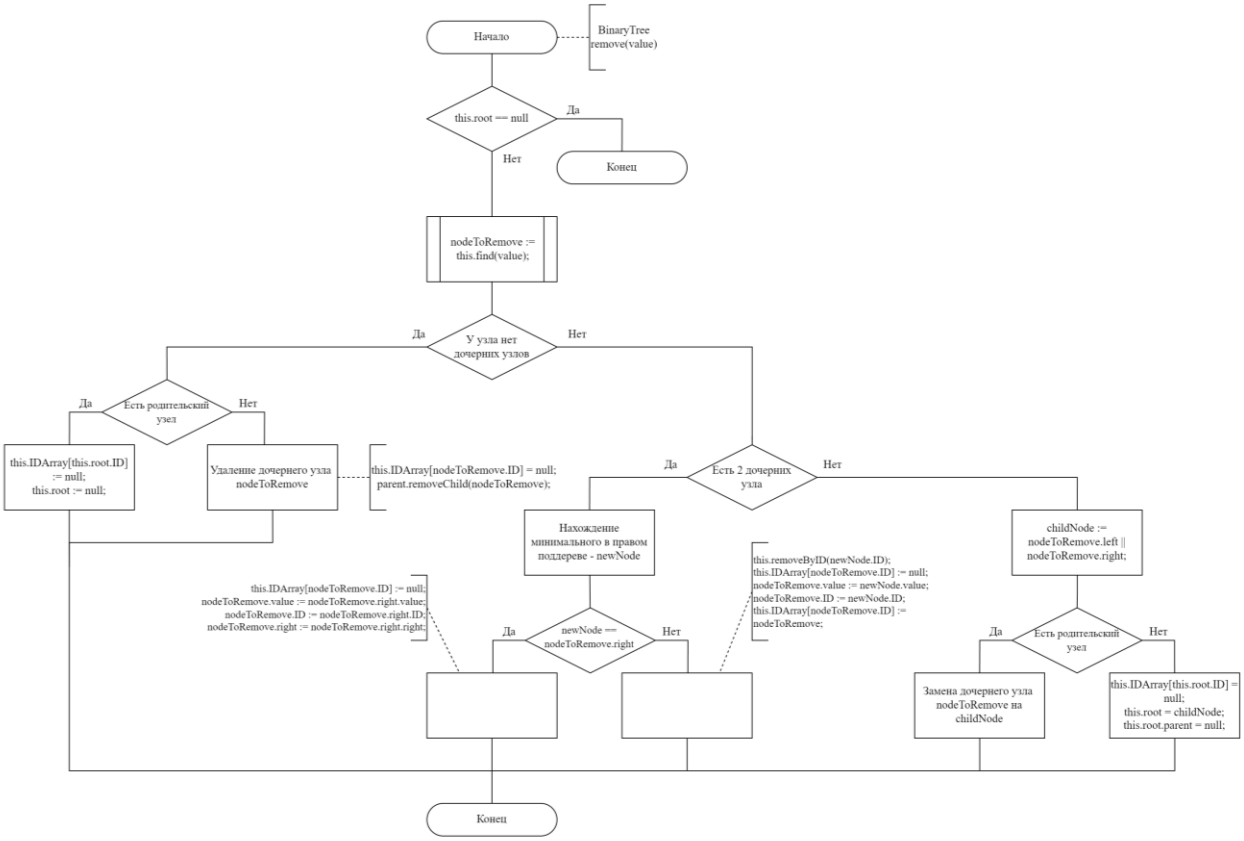

Рисунок 3 – Алгоритм метода удаления узла

2 Структурное описание разработки

2.1 Описание структуры приложения

Структура приложения с двумя классами для реализации бинарного дерева поиска выглядит следующим образом:

1. Класс BinaryTreeNode – представляет узел дерева и содержит следующие поля:

– ключ – значение ключа узла;

– данные – связанные с ключом данные;

– ссылка на левый потомок – ссылка на узел, являющийся левым потомком текущего узла;

– ссылка на правый потомок – ссылка на узел, являющийся правым потомком текущего узла;

– ссылка на родительский узел – ссылка на узел, являющийся родительским для текущего узла;

– счётчик вершин в поддереве.

Класс BinaryTreeNode содержит методы для работы с узлами дерева, например, для добавления, удаления, поиска и обхода узлов.

2. Класс BinaryTree – представляет само дерево и содержит следующие поля:

– корень дерева – ссылка на корневой узел дерева;

– количество узлов в дереве – переменная, которая хранит количество узлов в дереве;

– функция для сравнения значений вершин (функция - компаратор).

– массив с ссылками для доступа к узлу с конкретным номером.

Класс BinaryTree содержит методы для работы с деревом, например, для добавления, удаления, поиска и обхода узлов, а также для вывода дерева на экран в виде графической структуры.

Оба класса содержат конструкторы, деструкторы и другие вспомогательные методы для работы с данными. Взаимодействие между классами может осуществляется через вызов методов друг друга. Например, метод добавления нового узла в дерево может вызывать метод добавления нового узла в классе BinaryTreeNode, а метод поиска узла может вызывать метод поиска узла в классе BinaryTreeNode, начиная с корневого узла дерева.

2.2 Используемые алгоритмы

При реализации программы для работы с бинарным деревом поиска использовались следующие алгоритмы:

1. Алгоритм вставки нового узла в дерево – при добавлении нового узла в дерево необходимо определить, куда его следует поместить в соответствии с порядком ключей. Алгоритм может быть реализован рекурсивно или итеративно.

Процесс включения вершины состоит из трех частей данный процесс описан Никлаусом Виртом:

1. Прохода по пути поиска, пока не убедимся, что ключа в дереве нет.
2. Включения новой вершины в дерево и определения результирующих показателей балансировки.
3. «Отступления» назад по пути поиска и проверки в каждой вершине показателя сбалансированности. Если необходимо — балансировка.

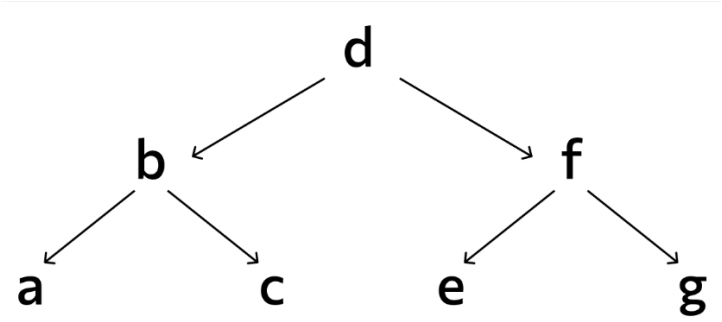

Рисунок 4 – Структура АВЛ-дерева

2. Алгоритм удаления узла из дерева – при удалении узла необходимо определить, какой узел должен заменить удаленный узел, чтобы сохранить свойства бинарного дерева поиска. Алгоритм может быть реализован рекурсивно или итеративно.

Для простоты опишем рекурсивный алгоритм удаления. Если вершина — лист, то удалим её и вызовем балансировку всех её предков в порядке от родителя к корню. Иначе найдём самую близкую по значению вершину в поддереве наибольшей высоты (правом или левом) и переместим её на место удаляемой вершины, при этом вызвав процедуру её удаления.

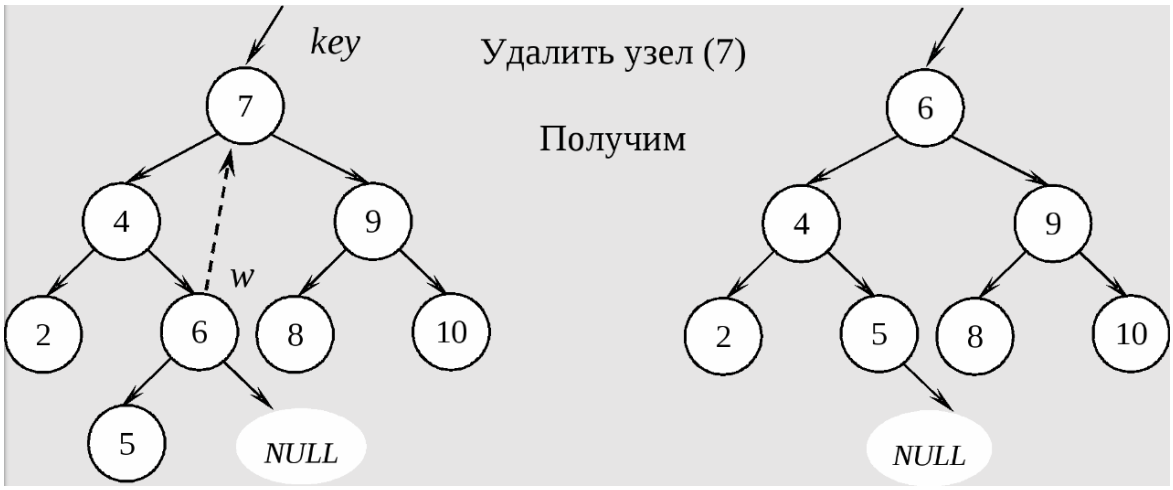

Рисунок 5 – Удаление узла из дерева

3. Алгоритм поиска узла в дереве – при поиске узла необходимо обойти дерево и найти узел с заданным ключом. Алгоритм может быть реализован рекурсивно или итеративно.

4. Балансировка двоичного дерева поиска – операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

Используются 4 типа вращений:

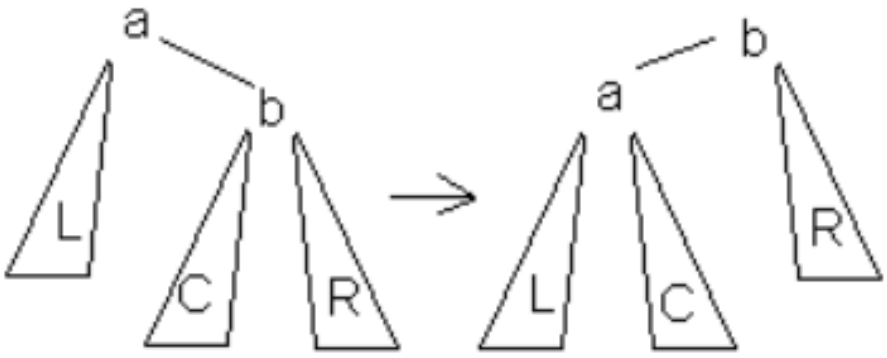

Рисунок 6 – Малое левое вращение

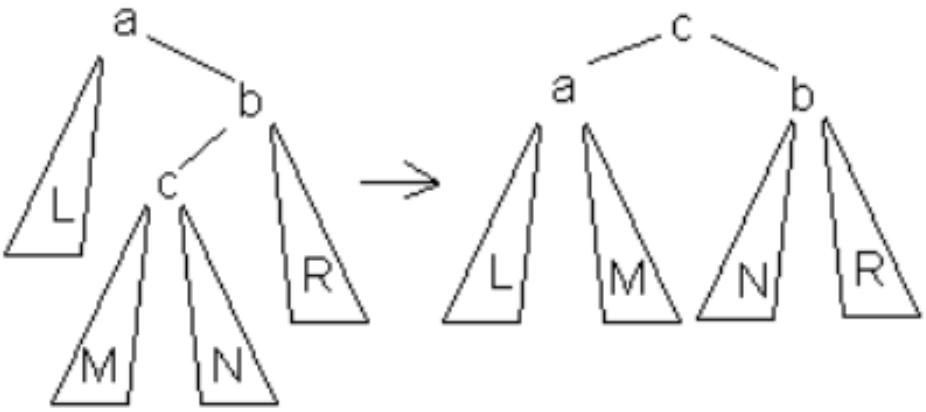

Рисунок 7 – Большое левое вращение

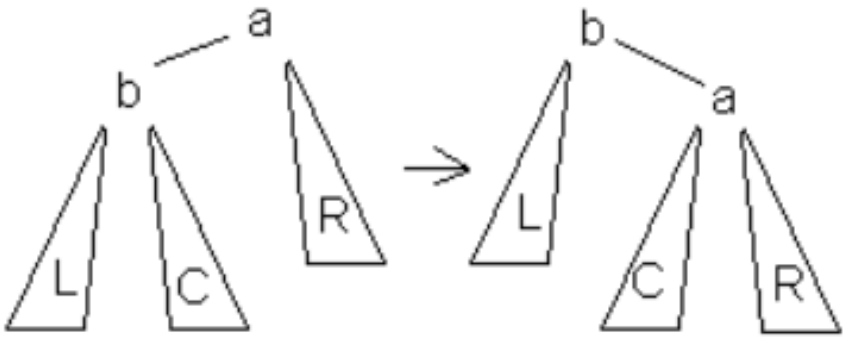

Рисунок 8 – Малое правое вращение

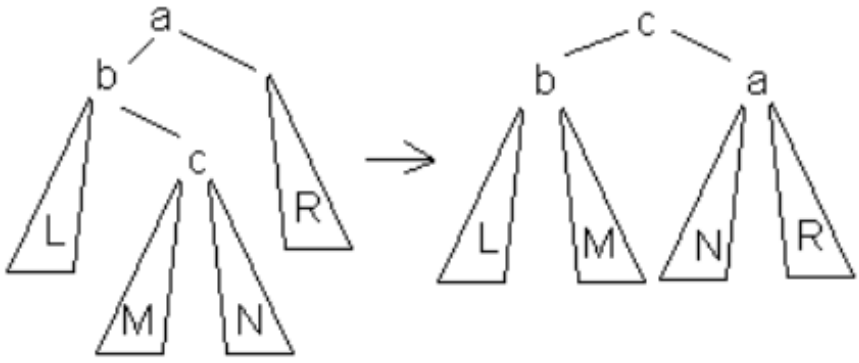

Рисунок 9 – Большое правое вращение

3 Функциональное описание структур данных и алгоритмов

3.1 Описание функционала приложения

Двоичное дерево поиска — это структура данных, которая представляет собой иерархическую структуру, состоящую из узлов и связей между ними. Каждый узел имеет значение и два потомка - левого и правого. Значения в левом поддереве меньше значения узла, а в правом - больше. Это позволяет быстро находить нужное значение, используя принцип бинарного поиска.

Одним из главных преимуществ двоичных деревьев является возможность быстрого добавления, удаления и поиска элементов.

Однако, при работе с деревьями возможны ситуации, когда дерево становится несбалансированным, что приводит к ухудшению производительности. Для решения этой проблемы используются алгоритмы балансировки дерева, такие как АВЛ-деревья и красно-черные деревья.

Приложение, разработанное в ходе выполнения курсового проекта, позволяет создать двоичное дерево поиска из чисел или массивов строк. В дерево можно добавить узел, новый узел добавляется в дерево путем поиска места для вставки. Если ключ нового узла меньше ключа корня текущего поддерева, то он помещается в левое поддерево, иначе - в правое поддерево. Если место для вставки уже занято и включён режим с повторением значений, то происходит вставка в левое поддерево.

Рисунок 10 – Интерфейс создания двоичного дерева поиска

Рисунок 11 – Интерфейс добавления нового узла в дерево

Также присутствует возможность удаления узла из дерева. Удаляемый узел может иметь одного или двух потомков. Если у удаляемого узла нет потомков, то он просто удаляется из дерева. Если у него есть один потомок, то этот потомок заменяет удаленный узел. Если у удаляемого узла есть два потомка, то нужно найти наименьший узел в правом поддереве и заменить им удаляемый узел.

При удалении узла осуществляется перестройка дерева в соответствии с актуальным количеством узлов дерева.

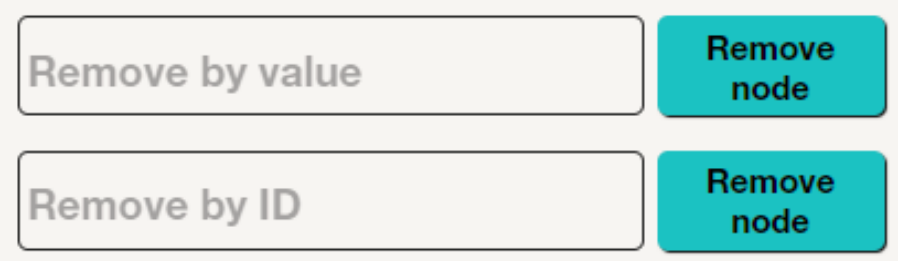

Рисунок 12 – Интерфейс удаления узла из дерева

Поиск узлов дерева происходит при удалении узла, а также в процессе выполнения других алгоритмов. Поиск узла осуществляется путем сравнения ключа текущего узла с искомым ключом. Если ключи равны, то узел найден. Если искомый ключ меньше ключа текущего узла, то поиск продолжается в левом поддереве, иначе - в правом поддереве.

Присутствует поддержка дубликатов узлов, а также функция автобалансировки дерева. Балансировка может быть произведена путем поворотов поддеревьев вокруг узлов, чтобы сохранить свойства бинарного дерева поиска и уменьшить высоту дерева.

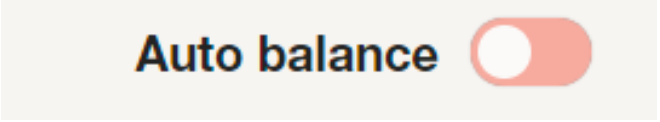

Рисунок 13 – Интерфейс функции автобалансировки

Реализация программы двоичного дерева поиска включает в себя создание класса, который содержит методы для вставки, удаления, поиска и балансировки дерева. Каждый узел представлен отдельным объектом класса, который содержит ключ и ссылки на левого и правого потомков.

Приложение включает в себя динамическую отрисовку при добавлении нового узла в дерева. За данную функцию отвечает отдельный переключатель в интерфейсе приложения.

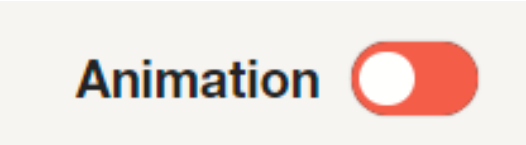

Рисунок 14 – Интерфейс динамической отрисовки

3.2 Описание структур данных и интерфейсов функций

Программными объектами выступают двоичное дерево и его узлы.

Двоичное дерево представлено классом BinaryTree содержащим в себе конструктор, поля и методы необходимые для работы с деревом.

```
class BinaryTree {
    root // ссылка на коневой узел дерева
    comparator // функция сравнения для значений узлов
    IDArray // массив для произвольного доступа к узлам
    constructor(comparator) {
        this.IDArray = [];
        this.comparator = comparator;
    }
    insert(value) {} // Вставка значения в дерево
    find(value) {} // Поиск узла по значению
    recalculateCounts() {} // Пересчёт значения количества вершин в поддереве от корня к листьям
    removeByID(id) {} // Удаление узла по ID
```

```
    remove(value) {} // Удаление узла по значению
    createSaveString() {} // Создание строки из дерева для последующего сохранения в файл
}
```

Узел дерева представлено классом BinaryTreeNode содержащим в себе конструктор, поля и методы необходимые для работы с деревом и его узлами.

```
class BinaryTreeNode {
    count = 1; // Количество узлов в поддереве
    left = null; // Ссылка на левый узел
    right = null; // Ссылка на правый узел
    parent = null; // Ссылка на родительский узел
    comparator; // Функция сравнения
    value; // Значение узла
    ID; // Порядковый номер узла
    DOMLink; // Ссылка на соответствующий узлу HTML тег
    constructor(value, comparator, ID) {
        this.value = value;
        this.comparator = comparator;
        this.ID = ID;
    }
    insert(value, ID) {} // Рекурсивная вставка в поддерево и присвоение ID
    find(value) {} // Поиск значения в поддереве
    findMin() {} // Поиск минимального элемента в поддереве
    removeChild(nodeToRemove) {} // Удаление дочернего узла
```

replaceChild(nodeToReplace, replacementNode) {} // Замена дочернего узла

recalculateAndRebalanceUp() {} // Пересчёт кол-ва узлов в поддереве и балансировка от листа к корню

recalculateUp() {} // Пересчёт кол-ва узлов в поддереваа от листа к корню

recalculateCountDown() {} // Пересчёт кол-ва узлов в поддереваа от корня к листьям

rotateLeft() {} // Малый левый поворот поддерева

rotateRight() {} // Малый правый поворот

bigRotateLeft() {} // Большой левый поворот

bigRotateRight() {} // Большой правый поворот

getHeight() {} // Получение глубины поддерева

balance() {} // Балансировка поддерева

createSaveString() {} // Создание строки из узла для последующего сохранения в файл

}

Взаимодействие с элементами интерфейса происходит посредствам обработки соответствующих событий, вызова функций и изменения состояний и режимов работы дерева.

4 Описание работы программы на конкретных примерах

4.1 Тестирование приложения

Приложение реализовано в виде сайта на языке гипертекстовой разметки HTML. Имеет рабочее поле, а также боковую панель с инструментами.

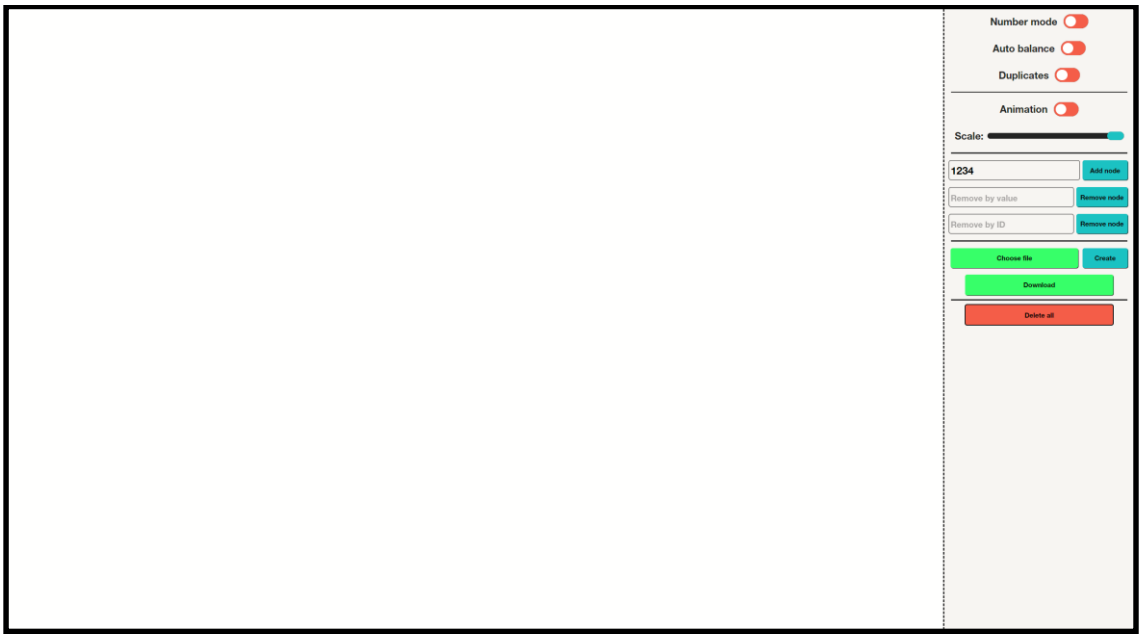

Рисунок 15 – Окно приложения

Для тестирования приложения создадим тестовые данные: набор ключей и соответствующих значений, которые будут добавлены в дерево.

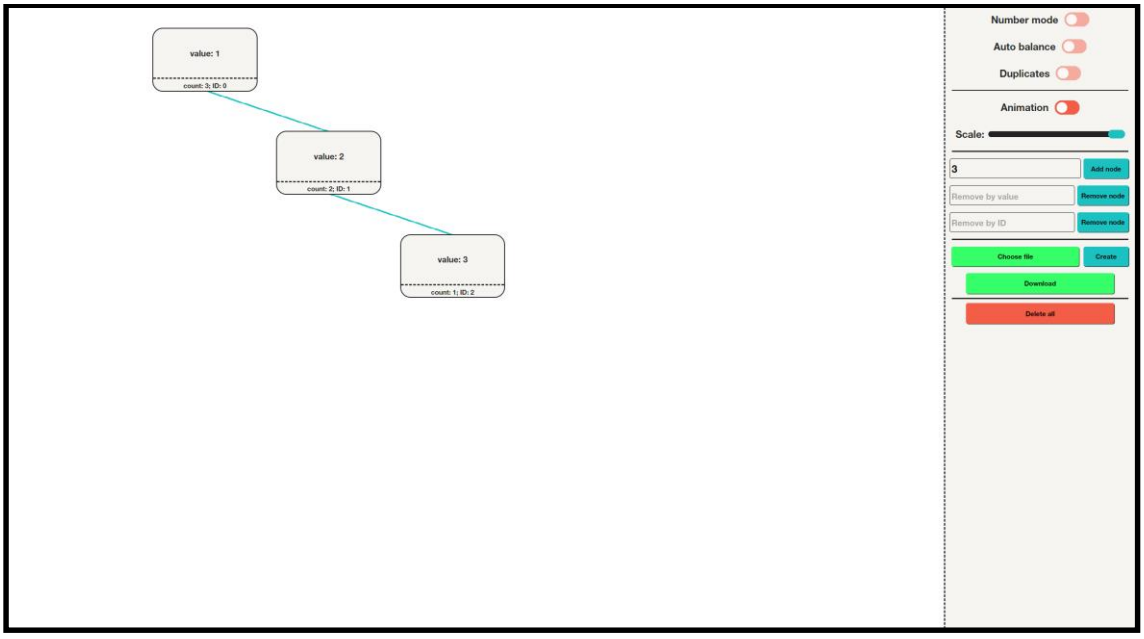

Рисунок 16 – Созданное дерево

Проверим корректность работы функции добавления узла и удаления узла из дерева. Для этого добавим еще один узел, после чего удалим один из ранее созданных.

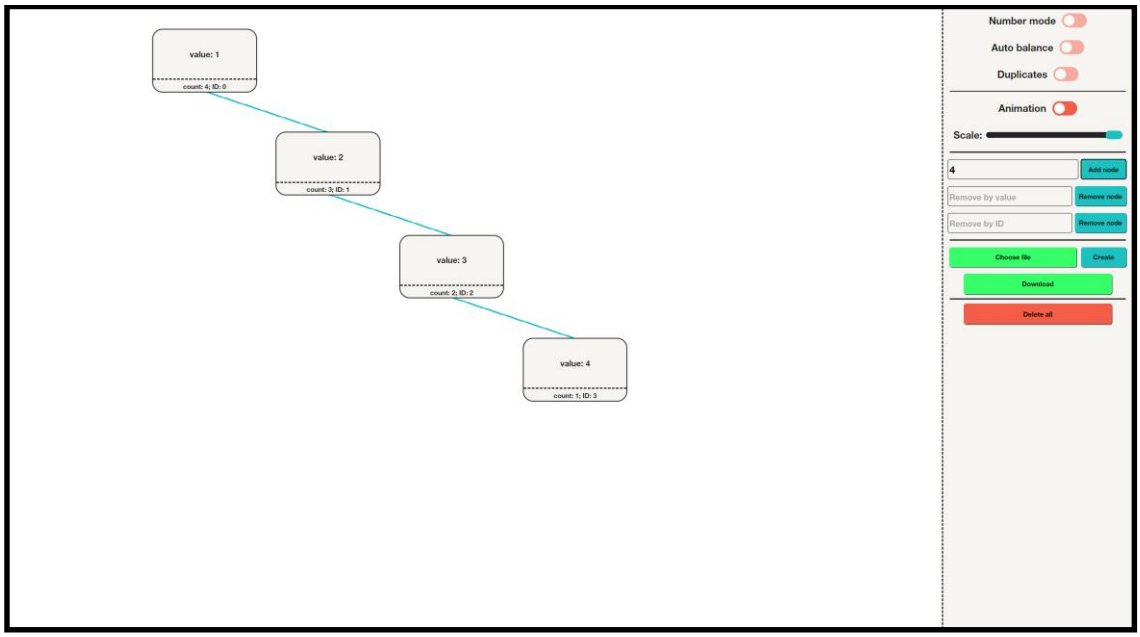

Рисунок 17 – Добавление узла в уже созданное дерево

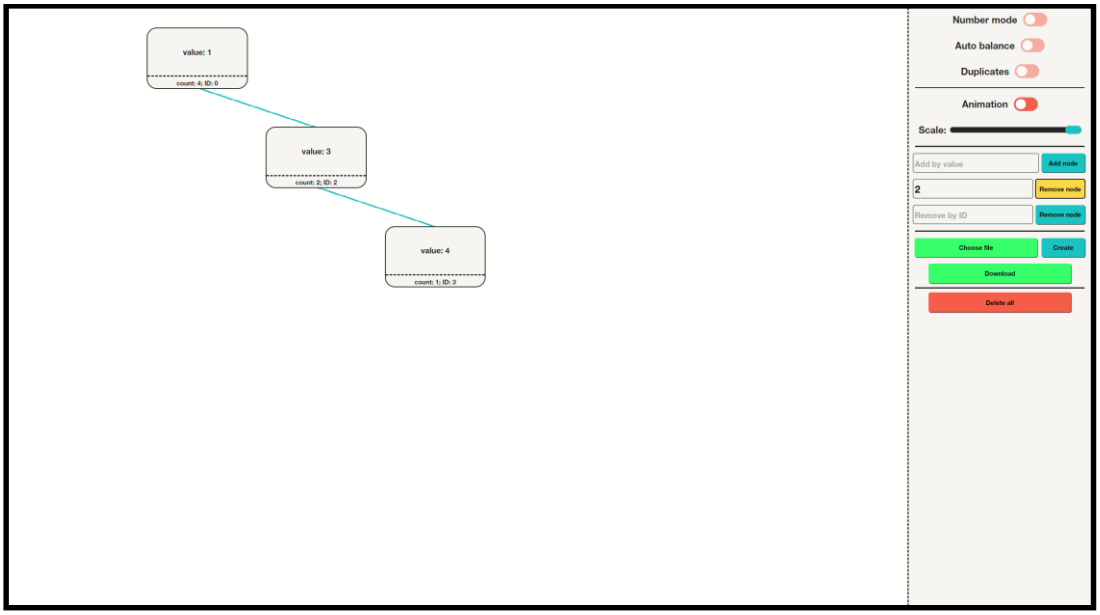

Рисунок 18 – Удаление узла по значению

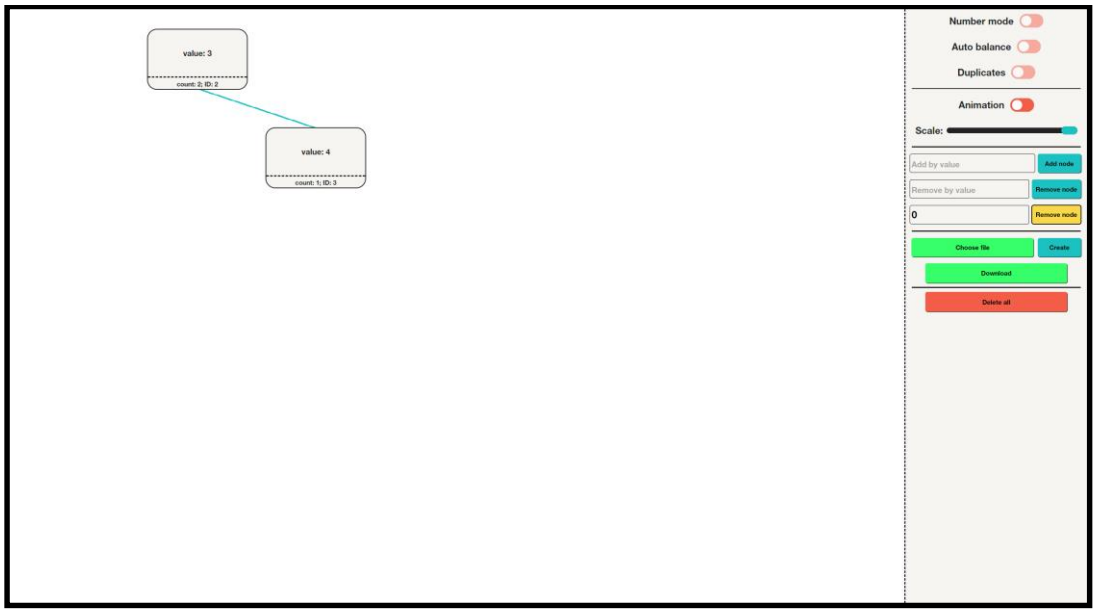

Рисунок 19 – Удаление узла по id

Как видно на снимках экрана функции приложения по добавлению, удалению узлов, а также по их поиску и балансировке дерева работают корректно.

В процессе работы на поток ввода от пользователя могут поступать некорректные входные данные. Приложение имеет индикацию и ведет себя в этом случае стабильно. Пример представлен на рисунках 17-18.

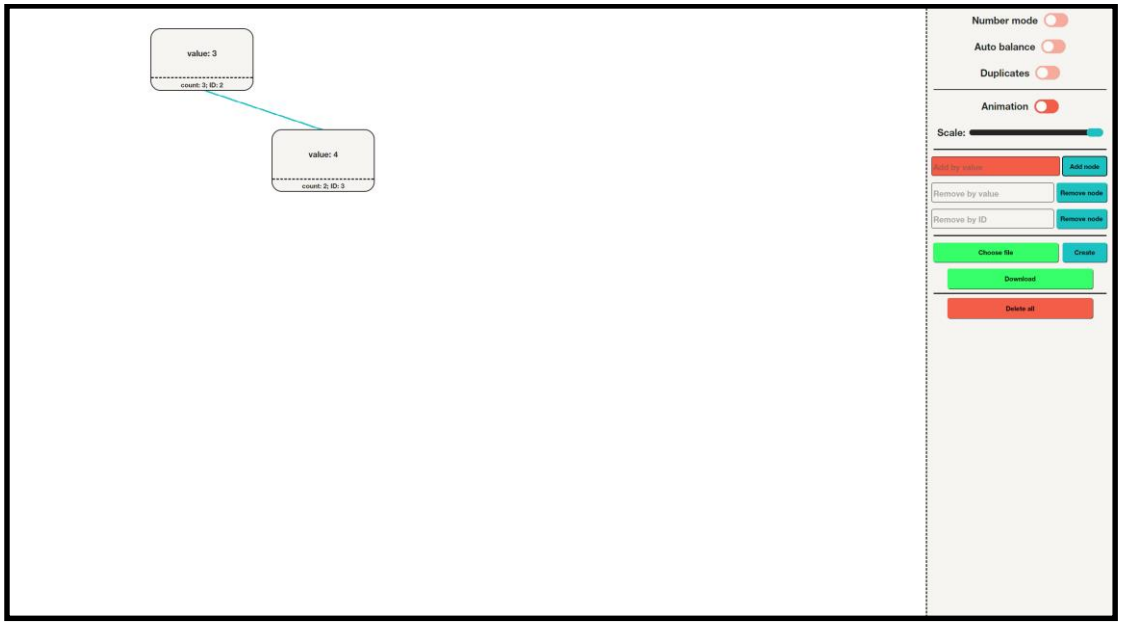

Рисунок 20 – Ввод пустого поля добавления узла

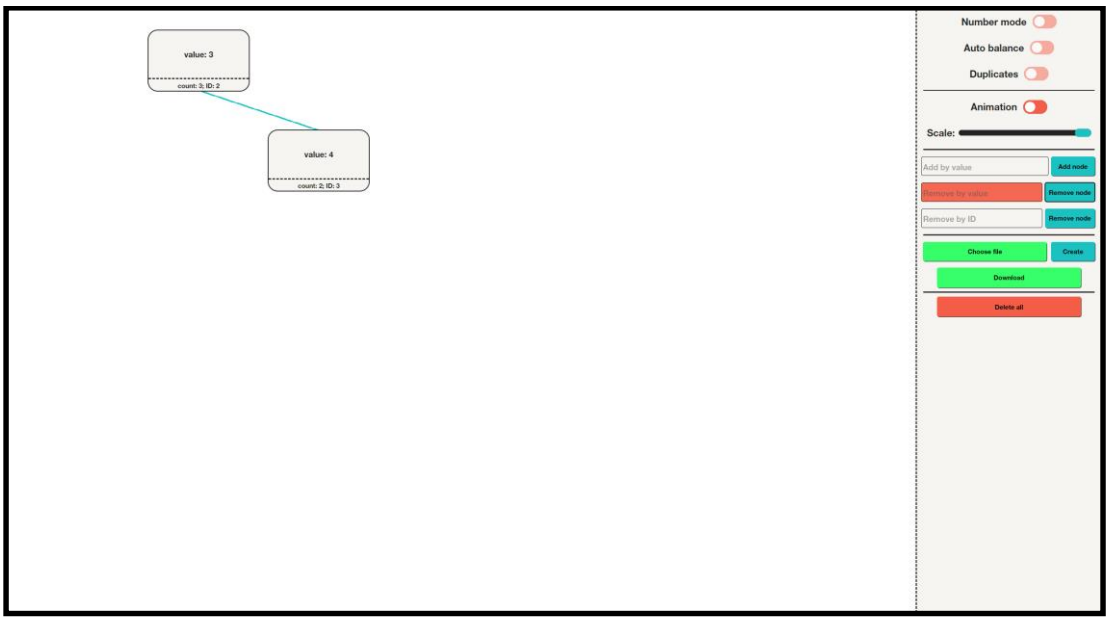

Рисунок 21 – Ввод пустого поля удаления узла

4.2 Выводы

Построение бинарного дерева поиска в приложении сопровождается динамически анимированной отрисовкой, что позволяет наглядно отслеживать работу функций приложения.

Приложение поддерживает добавление новых узлов в дерево, а также удаление узлов с дальнейшей перестройкой структуры бинарного дерева поиска. Удалить узел дерева возможно как по значению, так и по id узла.

Присутствует возможность скачивания созданного бинарного дерева в виде файла формата .txt.

Программа протестирована в исключительных и экспериментальных условиях, из чего можно сделать вывод о ее корректной работе, даже в случае подачи некорректных данных, система индикации при некорректных данных, подаваемых пользователем, присутствует.

ЗАКЛЮЧЕНИЕ

В процессе разработки программного обеспечения для решения конкретной задачи была изучена специфическая литература по теме проекта: грамотное и рациональное распределение задач, документация по среде разработки Visual Studio Code, документация по языку программирования JavaScript, гипертекстовой разметки HTML и каскадных стилей CSS. Была разработана структура данных, представляющая собой АВЛ-дерево, разработаны соответствующие алгоритмы работы с ним и графический интерфейс конечного пользователя. Так же была изучена JS библиотека для упрощённого рисования фигур в HTML документе.

Данное приложение может быть полезно во многих областях, включая базы данных, поиск и сортировку данных, алгоритмы машинного обучения и многие другие. При этом необходимо учитывать особенности реализации класса узла и методов, которые позволяют работать с деревом. Важно также понимать, что эффективность работы дерева зависит от правильной его балансировки и выбора оптимальных алгоритмов работы с данными.

Усовершенствовать данное приложения можно с помощью улучшения интерфейса, и добавления нового функционала, например, добавление большего числа алгоритмов для динамической демонстрации.

Опробовать приложение можно по ссылке https://kanzu32.github.io/BinaryTree/

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. "ГОСТ 7.32-2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления" (введен в действие Приказом Росстандарта от 24.10.2017 N 1494-ст)
2. Макмилан, М. Структуры данных и алгоритмы в JavaScript. – O'Reilly, 2014. – 246 с.
3. Вирт, Н. Алгоритмы и структуры данных. – М.: ДМК Пресс, 2016 – 272 с.
4. Фримен, Э Изучаем программирование на JavaScript». – СПб.: Издательство Питер, 2022. – 640с.
5. Фримен, Э Изучаем HTML, XHTML и CSS (Head First). – СПб.: Издательство Питер, 2022. – 720с.
6. Современный учебник JavaScript : сайт. – URL: https://learn.javascript.ru (дата обращения: 15.03.2023)

ПРИЛОЖЕНИЕ А

Текст программы

//script.js

```
function comparator(a, b) { // Функция для сравнения данных из
узлов дерева
    return (b<a) - (a<b);
}
class BinaryTree {
    root
    comparator
    IDArray
    constructor(comparator) {
        this.IDArray = [];
        this.comparator = comparator;
    }
    insert(value) { // Вставка значения в дерево
        if (this.root == null) {
            this.root = new BinaryTreeNode(value,
this.comparator, this.IDArray.length);
            this.IDArray.push(this.root);
            if (animation) newNodesTrace.push([this.root,
"marked-green"]);
            setTypeChange(false);
            setAutoBalanceChange(false);
            setDuplicateChange(false);
            return this.root;
        } else {
            let node = this.root.insert(value,
this.IDArray.length);
            if (node) this.IDArray.push(node);
            return node;
        }
    }
    find(value) { // Поиск узла по значению
        return this.root.find(value);
    }
    recalculateCounts() { // Пересчёт значения количества вершин
в поддереве от корня к листьям
        this.root.recalculateCountDown();
    }
    removeByID(id) { // Удаление узла по ID
        if (this.root == null) return;
        const nodeToRemove = this.IDArray[id];
        animationTrace.push([nodeToRemove.DOMLink, "marked-
red"]);
        if (nodeToRemove == null) {
            console.log("Tree with id " + id + " is not
found.");
            return false;
        }
        const parent = nodeToRemove.parent;
```

```
        if (!nodeToRemove.left && !nodeToRemove.right) {
            if (parent) {
                this.IDArray[nodeToRemove.ID] = null;
                parent.removeChild(nodeToRemove);
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
            } else {
                this.IDArray[this.root.ID] = null;
                this.root = null;
                setTypeChange(true);
                setAutoBalanceChange(true);
                setDuplicateChange(true)
            }
        } else if (nodeToRemove.left && nodeToRemove.right) {
            const newNode = nodeToRemove.right.findMin();
            if (newNode === nodeToRemove.right) {
                this.IDArray[nodeToRemove.ID] = null;
                nodeToRemove.value = nodeToRemove.right.value;
                nodeToRemove.ID = nodeToRemove.right.ID;
                nodeToRemove.right = nodeToRemove.right.right;
                this.IDArray[nodeToRemove.ID] = nodeToRemove;
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
            } else {
                this.removeByID(newNode.ID);
                this.IDArray[nodeToRemove.ID] = null;
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
                nodeToRemove.value = newNode.value;
                nodeToRemove.ID = newNode.ID;
                this.IDArray[nodeToRemove.ID] = nodeToRemove;
            }
        } else {
            const childNode = nodeToRemove.left ||
nodeToRemove.right;
            if (animation)
animationTrace.push([childNode.DOMLink, "marked-green"]);
            if (parent) {
                this.IDArray[nodeToRemove.ID] = null;
                parent.replaceChild(nodeToRemove, childNode);
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
            } else {
                this.IDArray[this.root.ID] = null;
                this.root = childNode;
                this.root.parent = null;
            }
        }
```

```
        return true;
    }
    remove(value) { // Удаление узла по значению
        if (this.root == null) return;
        const nodeToRemove = this.find(value);

        if (nodeToRemove == null) {
            console.log("Tree with value " + value + " is not
found.");
            return false;
        }
        const parent = nodeToRemove.parent;
        if (!nodeToRemove.left && !nodeToRemove.right) {
            if (parent) {
                this.IDArray[nodeToRemove.ID] = null;
                parent.removeChild(nodeToRemove);
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
            } else {
                this.IDArray[this.root.ID] = null;
                this.root = null;
                setTypeChange(true);
                setAutoBalanceChange(true);
                setDuplicateChange(true)
            }
        } else if (nodeToRemove.left && nodeToRemove.right) {
            const newNode = nodeToRemove.right.findMin();
            if (newNode === nodeToRemove.right) {
                this.IDArray[nodeToRemove.ID] = null;
                nodeToRemove.value = nodeToRemove.right.value;
                nodeToRemove.ID = nodeToRemove.right.ID;
                nodeToRemove.right = nodeToRemove.right.right;
                this.IDArray[nodeToRemove.ID] = nodeToRemove;
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
            } else {
                this.removeByID(newNode.ID);
                this.IDArray[nodeToRemove.ID] = null;
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
                nodeToRemove.value = newNode.value;
                nodeToRemove.ID = newNode.ID;
                this.IDArray[nodeToRemove.ID] = nodeToRemove;
            }
        } else {
            const childNode = nodeToRemove.left ||
nodeToRemove.right;
            if (animation)
animationTrace.push([childNode.DOMLink, "marked-green"]);
```

```
            if (parent) {
                this.IDArray[nodeToRemove.ID] = null;
                parent.replaceChild(nodeToRemove, childNode);
                if (autoBalance)
nodeToRemove.recalculateAndRebalanceUp();
                else nodeToRemove.recalculateUp();
            } else {
                this.IDArray[this.root.ID] = null;
                this.root = childNode;
                this.root.parent = null;
            }
        }
        return true;
    }

    createSaveString() { // Создание строки из дерева для
последующего сохранения в файл
        let res = (+autoBalance).toString() + "|" +
(+intMode).toString() + "|" + (+duplicate).toString();
        if (this.root) res += "|" +
this.root.createSaveString();
        return res;
    }
}

class BinaryTreeNode {
    count = 1;
    left = null;
    right = null;
    parent = null;
    comparator;
    value;
    ID;
    DOMLink;
    constructor(value, comparator, ID) {
        this.value = value;
        this.comparator = comparator;
        this.ID = ID;
    }
    insert(value, ID) { // Рекурсивная вставка в поддерево и
присвоение ID
        this.count++;
        if (animation) animationTrace.push([this.DOMLink,
"marked-yellow"]);
        if ((duplicate && this.comparator(value, this.value) ==
0) || (this.comparator(value, this.value) < 0)) {
            if (this.left) {
                let node = this.left.insert(value, ID);
                if (autoBalance) this.balance();
                return node;
            }
```

```
            this.left = new BinaryTreeNode(value,
this.comparator, ID);
            if (animation) newNodesTrace.push([this.left,
"marked-green"]);
            this.left.parent = this;
            if (autoBalance) this.balance();
            return this.left;
        }
        if (this.comparator(value, this.value) > 0) {
            if (this.right) {
                let node = this.right.insert(value, ID);
                if (autoBalance) this.balance();
                return node;
            }
            this.right = new BinaryTreeNode(value,
this.comparator, ID);
            if (animation) newNodesTrace.push([this.right,
"marked-green"]);
            this.right.parent = this;
            if (autoBalance) this.balance();
            return this.right;
        }
        this.count--;
        return null;
    }
    find(value) { // Поиск значения в поддереве
        if (this.comparator(this.value, value) === 0) {
            if (animation) animationTrace.push([this.DOMLink,
"marked-red"]);
            return this;
        }
        if (this.comparator(this.value, value) > 0 && this.left)
{
            if (animation) animationTrace.push([this.DOMLink,
"marked-yellow"]);
            return this.left.find(value);
        }
        if (this.comparator(this.DOMLink.value, value) <= 0 &&
this.right) {
            if (animation) animationTrace.push([this.DOMLink,
"marked-yellow"]);
            return this.right.find(value);
        }
        return null;
    }
    findMin() { // Поиск минимального элемента в поддереве
        if (!this.left) {
            if (animation) animationTrace.push([this.DOMLink,
"marked-green"]);
            return this;
        }
```

```
        if (animation) animationTrace.push([this.DOMLink,
"marked-yellow"]);
        return this.left.findMin();
    }
    removeChild(nodeToRemove) { // Удадение дочернего узла
        if (this.left && this.left === nodeToRemove) {
            this.left = null;
            return true;
        }
        if (this.right && this.right === nodeToRemove) {
            this.right = null;
            return true;
        }
        return false;
    }
    replaceChild(nodeToReplace, replacementNode) { // Замена
дочернего узла
        if (!nodeToReplace || !replacementNode) {
            return false;
        }
        if (this.left && this.left === nodeToReplace) {
            this.left = replacementNode;
            replacementNode.parent = this;
            return true;
        }
        if (this.right && this.right === nodeToReplace) {
            this.right = replacementNode;
            replacementNode.parent = this;
            return true;
        }
        return false;
    }
    recalculateAndRebalanceUp() { // Пересчёт кол-ва узлов в
поддереве и балансировка от листа к корню
        this.count = 1;
        if (this.left) {
            this.count += this.left.count;
        }
        if (this.right) {
            this.count += this.right.count;
        }
        this.balance();
        if (this.parent) {
            this.parent.recalculateAndRebalanceUp();
        }
    }
    recalculateUp() { // Пересчёт кол-ва узлов в поддеревеа от
листа к корню
        this.count = 1;
        if (this.left) {
            this.count += this.left.count;
        }
```

```
        if (this.right) {
            this.count += this.right.count;
        }
    }
    recalculateCountDown() { // Пересчёт кол-ва узлов в
поддеревеа от корня к листьям
        this.count = 1;
        if (this.left) {
            this.count += this.left.recalculateCountDown();
        }
        if (this.right) {
            this.count += this.right.recalculateCountDown();
        }
        return this.count;
    }
    rotateLeft() { // Малый левый поворот поддерева
        let b = this.right;
        this.right = b.left;
        if (b.left) {
            b.left.parent = this;
        }
        b.left = this;
        b.parent = this.parent;
        if (this.parent) {
            this.parent.replaceChild(this, b);
        } else {
            tree.root = b;
        }
        this.parent = b;
        if (animation) newNodesTrace.push([b, "marked-yellow"]);
        this.recalculateCountDown();
    }
    rotateRight() { // Малый правый поворот
        let b = this.left;
        this.left = b.right;
        if (b.right) {
            b.right.parent = this;
        }
        b.right = this;
        b.parent = this.parent;
        if (this.parent) {
            this.parent.replaceChild(this, b);
        } else {
            tree.root = b;
        }
        this.parent = b;
        if (animation) newNodesTrace.push([b, "marked-yellow"]);
        this.recalculateCountDown();
    }
    bigRotateLeft() { // Большой левый поворот
        this.right.rotateRight()
```

```
        this.rotateLeft()
    }
    bigRotateRight() { // Большой правый поворот
        this.left.rotateLeft()
        this.rotateRight()
    }
    getHeight() { // Получение глубины поддерева
        if (!this.left && !this.right) {
            return 1;
        } else if (this.left && this.right) {
            return Math.max(this.left.getHeight(),
this.right.getHeight()) + 1;
        } else if (!this.left) {
            return this.right.getHeight() + 1;
        } else {
            return this.left.getHeight() + 1;
        }
    }
    balance() { // Балансировка поддерева
        let rightHeight = 0
        let leftHeight = 0
        if (this.right) rightHeight = this.right.getHeight();
        if (this.left) leftHeight = this.left.getHeight();
        let bal = rightHeight - leftHeight;

        if (bal >= 2) { //left
            if (animation) newNodesTrace.push([this, "marked-
red"]);
            let subRightHeight = 0;
            let subLeftHeight = 0;
            if (this.right.right) subRightHeight =
this.right.right.getHeight();
            if (this.right.left) subLeftHeight =
this.right.left.getHeight();

            if (subLeftHeight <= subRightHeight)
this.rotateLeft();
            else this.bigRotateLeft();

        } else if (bal <= -2) { //right
            if (animation) newNodesTrace.push([this, "marked-
red"]);
            let subRightHeight = 0;
            let subLeftHeight = 0;
            if (this.left.right) subRightHeight =
this.left.right.getHeight();
            if (this.left.left) subLeftHeight =
this.left.left.getHeight();

            if (subRightHeight <= subLeftHeight)
this.rotateRight();
            else this.bigRotateRight();
```

```
        }
    }
    createSaveString() { // Создание строки из узла для
последующего сохранения в файл
        if (this.left) saveQueue.push(this.left);
        if (this.right) saveQueue.push(this.right);
        if (saveQueue.length == 0) {
            if (intMode) return this.value.toString();
            else return this.value.join(" ");
        } else {
            if (intMode) return this.value.toString() + "|" +
saveQueue.shift().createSaveString();
            else return this.value.join(" ") + "|" +
saveQueue.shift().createSaveString();
        }
    }
}
// Управление и навигация по дереву
const view = document.getElementById("view");
const content = document.getElementById("content");
let pos = { top: 0, left: 0, x: 0, y: 0 };

const mouseDownHandler = function (e) {
    view.style.cursor = 'grabbing';
    view.style.userSelect = 'none';
    pos = {
        // The current scroll
        left: view.scrollLeft,
        top: view.scrollTop,
        // Get the current mouse position
        x: e.clientX,
        y: e.clientY,
    };
    document.addEventListener('mousemove', mouseMoveHandler);
    document.addEventListener('mouseup', mouseUpHandler);
};
const mouseMoveHandler = function (e) {
    const dx = e.clientX - pos.x;
    const dy = e.clientY - pos.y;

    view.scrollTop = pos.top - dy;
    view.scrollLeft = pos.left - dx;
};
const mouseUpHandler = function () {
    document.removeEventListener('mousemove', mouseMoveHandler);
    document.removeEventListener('mouseup', mouseUpHandler);

    view.style.cursor = 'grab';
    view.style.removeProperty('user-select');
};
view.addEventListener('mousedown', mouseDownHandler);
```

```
// Функции отрисовки дерева
function drawNode(node, x, y) {
    let block = document.createElement('div');
    block.classList.add('block');
    block.style.left = x+"px";
    block.style.top = y+"px";
    block.style.height = blockHeight+"px";
    block.style.width = blockWidth+"px";
    let blockVal = document.createElement("div");
    blockVal.classList.add('block-value');
    let blockInfo = document.createElement("div");
    blockInfo.classList.add('block-info');
    let stringValue;
    if (intMode) stringValue = node.value.toString();
    else stringValue = node.value.join(" ");
    if (stringValue.length > 45) blockVal.style.fontSize =
"16px";
    if (stringValue.length > 70) stringValue =
stringValue.slice(0, 69) + "...";
    blockVal.textContent = "value: " + stringValue;
    blockInfo.textContent = "count: " + node.count + "; ID: " +
node.ID;
    block.appendChild(blockVal);
    block.appendChild(blockInfo);
    content.appendChild(block);
    node.DOMLink = block;
}
function drawTree(tree) {
    content.innerHTML = '';
    jg = new jsGraphics(content);
    jg.setColor("#1ac2c2");
    jg.setStroke(3);
    let count = 1;
    if (tree.root == null) return;
    if (tree.root.left != null) count = tree.root.left.count;
    let x = xTreeOffset+(xOffset*count);
    let y = yTreeOffset;
    if (x > maxX) {maxX = x};
    if (y > maxY) {maxY = y};
    drawNode(tree.root, x, y);
    if (tree.root.left != null) drawLeft(tree.root.left, x, y);
    if (tree.root.right != null) drawRight(tree.root.right, x,
y);
    jg.paint();
    content.style.height = maxY+blockHeight+6+yTreeOffset+"px";
    content.style.width = maxX+blockWidth+6+xTreeOffset+"px";
}
function drawLeft(node, parentX, parentY) {
    let count = 0;
    if (node.right != null) count = node.right.count;
    let x = parentX - xOffset - (count * xOffset);
    let y = parentY + yOffset;
```

```
    if (x > maxX) {maxX = x};
    if (y > maxY) {maxY = y};
    jg.drawLine(parentX+blockWidth/2, parentY+blockHeight,
x+blockWidth/2, y);
    drawNode(node, x, y);
    if (node.left != null) drawLeft(node.left, x, y);
    if (node.right != null) drawRight(node.right, x, y);
}
function drawRight(node, parentX, parentY) {
    let count = 0;
    if (node.left != null) count = node.left.count;
    let x = parentX + xOffset + (count * xOffset);
    let y = parentY + yOffset;
    if (x > maxX) {maxX = x};
    if (y > maxY) {maxY = y};
    jg.drawLine(parentX+blockWidth/2, parentY+blockHeight,
x+blockWidth/2, y);
    drawNode(node, x, y);
    if (node.left != null) drawLeft(node.left, x, y);
    if (node.right != null) drawRight(node.right, x, y);
}
// Функции для обработки взаимодействия с интерфейсом
function addNode() {
    stopAnimation();
    let input = document.getElementById("new-node-input");
    let value = input.value;
    let res;
    if (intMode && isNaN(value) || value == '') {
        input.classList.add("wrong-input-animation");
        input.onanimationend = () =>
{input.classList.remove("wrong-input-animation");};
    } else if (intMode) {
        res = tree.insert(parseInt(value, 10));
        if (res) {
            if (animation) animate();
            else drawTree(tree);
        } else {
            input.classList.add("wrong-input-animation");
            input.onanimationend = () =>
{input.classList.remove("wrong-input-animation");};
        }
    } else {
        res = tree.insert(value.split(" "));
        if (res) {
            if (animation) animate();
            else drawTree(tree);
        } else {
            input.classList.add("wrong-input-animation");
            input.onanimationend = () =>
{input.classList.remove("wrong-input-animation");};
        }
    }
```

```
}
function removeNode() {
    stopAnimation();
    let input = document.getElementById("remove-node-input");
    let value = input.value;
    if (value == "" || (intMode && isNaN(value)) || tree.root ==
null) {
        input.classList.add("wrong-input-animation");
        input.onanimationend = () =>
{input.classList.remove("wrong-input-animation");};
    } else if (intMode) {
        tree.remove(parseInt(value, 10));
        if (animation) animate();
        else drawTree(tree);
    } else {
        tree.remove(value.split(" "));
        if (animation) animate();
        else drawTree(tree)
    }
}
function removeNodeByID(id = -1) {
    stopAnimation();
    if (id < 0) {
        let input = document.getElementById("remove-id-node-
input");
        id = parseInt(input.value, 10);
        if (id.isNaN || id < 0 || tree.IDArray[id] == null) {
            input.classList.add("wrong-input-animation");
            input.onanimationend = () =>
{input.classList.remove("wrong-input-animation");};
        } else {
            tree.removeByID(id);
            if (animation) animate();
            else drawTree(tree)
        }
    } else if (tree.IDArray[id] != null) {
        tree.removeByID(id);
        if (animation) animate();
        else drawTree(tree)
    }
}
dataTypeSwitch = document.getElementById("data-type-input");
dataTypeSwitch.addEventListener("change", ()=>{
    if (!tree.root) {
        intMode = dataTypeSwitch.checked;
    }
});
function setTypeChange(val) {
    dataTypeSwitch.disabled = !val;
    dataTypeSwitch.parentElement.style.opacity = 0.5 + val*0.5;
};
balanceSwitch = document.getElementById("balance-input");
```

```
balanceSwitch.addEventListener("change", ()=>{
    if (!tree.root) {
        autoBalance = balanceSwitch.checked;
    }
});
function setAutoBalanceChange(val) {
    balanceSwitch.disabled = !val;
    balanceSwitch.parentElement.style.opacity = 0.5 + val*0.5;
}
duplicateSwitch = document.getElementById("duplicate-input");
duplicateSwitch.addEventListener("change", ()=>{
    if (!tree.root) {
        duplicate = duplicateSwitch.checked;
    }
});
function setDuplicateChange(val) {
    duplicateSwitch.disabled = !val;
    duplicateSwitch.parentElement.style.opacity = 0.5 + val*0.5;
}
scaleSlider = document.getElementById("scale-input");
scaleSlider.value = 1;
scaleSlider.addEventListener("change", () => {
    content.style.scale = scaleSlider.value;
});
animationSwitch = document.getElementById("animation-input");
animationSwitch.addEventListener("change", ()=>{
    stopAnimation();
    animation = animationSwitch.checked;
});
function download() {
    stopAnimation()
    let text = tree.createSaveString();
    let element = document.createElement('a');
    element.setAttribute('href', 'data:text/plain;charset=utf-
8,' + encodeURIComponent(text));
    element.setAttribute('download', "tree.txt");

    element.style.display = 'none';
    document.body.appendChild(element);

    element.click();

    document.body.removeChild(element);
};
let fileInput = document.getElementById("file-input");

fileInput.addEventListener("change", () => {
    let text = fileInput.files[0].name;
    if (text.length > 15) text = text.slice(0, 14) + "...";
    document.querySelector("#input-file-wrapper >
p").textContent = text;
});
```

```
function load() {
    stopAnimation()
    let fr = new FileReader();
    let file = fileInput.files[0];
    fr.readAsText(file);
    fr.onload = function () {
        let str = fr.result;
        let input = str.split("|");
        if (!isNaN(parseInt(input[0], 10)) &&
!isNaN(parseInt(input[1], 10)) && !isNaN(parseInt(input[2],
10))) {
            autoBalance = !!parseInt(input[0], 10);
            intMode = !!parseInt(input[1], 10);
            duplicate = !!parseInt(input[2], 10);
            balanceSwitch.checked = autoBalance;
            dataTypeSwitch.checked = intMode;
            duplicateSwitch.checked = duplicate;
        } else {
            fileInput.classList.add("wrong-input-animation");
            fileInput.onanimationend = () =>
{fileInput.classList.remove("wrong-input-animation");};
            return;
        }
        deleteAll();
        for (let i = 3; i < input.length; i++) {
            if (intMode && isNaN(parseInt(input[i], 10))) {
                fileInput.classList.add("wrong-input-
animation");
                fileInput.onanimationend = () =>
{fileInput.classList.remove("wrong-input-animation");};
                deleteAll();
                return;
            }
            let res;
            if (intMode) {
                res = tree.insert(parseInt(input[i], 10));
            } else {
                res = tree.insert(input[i].split(" "));
            }
            if (res) {
                drawTree(tree);
            } else {
                fileInput.classList.add("wrong-input-
animation");
                fileInput.onanimationend = () =>
{fileInput.classList.remove("wrong-input-animation");};
                deleteAll();
            }
        }
    };
}
```

```
function deleteAll() {
    tree = new BinaryTree(comparator);
    autoBalance = balanceSwitch.checked;
    intMode = dataTypeSwitch.checked;
    duplicate = duplicateSwitch.checked;
    setTypeChange(true);
    setAutoBalanceChange(true);
    setDuplicateChange(true);
    drawTree(tree);
}
var treeDrawed = false;
function animate() {
    if (animationTrace.length > 0) {
        let item = animationTrace[0];
        item[0].classList.add(item[1]);
    } else if (newNodesTrace.length > 0) {
        drawTree(tree);
        treeDrawed = true;
        let item = newNodesTrace[0];
        item[0].DOMLink.classList.add(item[1]);
    }
    treeDrawed = false;
    animationTimer = setInterval(()=>{
        if (animationTrace.length > 0) {
            let item = animationTrace.shift();
            item[0].classList.remove(item[1]);
        }
        if (animationTrace.length > 0) {
            let item = animationTrace[0];
            item[0].classList.add(item[1]);
        } else if (!treeDrawed){
            drawTree(tree);
            treeDrawed = true;
            if (newNodesTrace.length > 0) {
                let item = newNodesTrace[0];
                item[0].DOMLink.classList.add(item[1]);
            }
        } else if (newNodesTrace.length > 0) {
            let item = newNodesTrace.shift();
            item[0].DOMLink.classList.remove(item[1]);
            if (newNodesTrace.length > 0) {
                let item = newNodesTrace[0];
                item[0].DOMLink.classList.add(item[1]);
            }
        } else {
            stopAnimation();
        }
    }, animationTime);
}
function stopAnimation() {
    clearInterval(animationTimer);
```

```
    if (animationTrace.length > 0)
{animationTrace[0][0].classList.remove(animationTrace[0][1]);
    } else if (newNodesTrace.length > 0) {

newNodesTrace[0][0].DOMLink.classList.remove(newNodesTrace[0][1]
);
    }
    animationTrace = [];
    newNodesTrace = [];
    drawTree(tree);
}
// Инициализация дерева и его параметров
var jg;
var autoBalance = balanceSwitch.checked;
var intMode = dataTypeSwitch.checked;
var duplicate = duplicateSwitch.checked;
var animation = animationSwitch.checked;
var animationTrace = [];
var newNodesTrace = [];
var animationTime = 500;
var animationTimer;
var xTreeOffset = 50;
var yTreeOffset = 50;
var xOffset = 300;
var yOffset = 250;
var blockWidth = 250;
var blockHeight = 150;
var maxX = 0;
var maxY = 0;
let tree = new BinaryTree(comparator);
var saveQueue = [];
// Начальная отрисовка дерева
drawTree(tree);
```

//index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Binary Tree</title>
    <link rel="stylesheet" href="style.css">
    <link href="https://fonts.cdnfonts.com/css/helvetica-neue-9"
rel="stylesheet">
    <script type="text/javascript"
src="wz_jsgraphics.js"></script>
</head>
<body>
    <div id="view">
        <div id="content"></div>
    </div>
    <div id="sidebar">
```

```
        <div class="bar-item">
            <p class="text-hint">Number mode</p>
            <label class="switch">
                <input type="checkbox" id="data-type-input">
                <span class="slider round"></span>
            </label>
        </div>
        <div class="bar-item">
            <p class="text-hint">Auto balance</p>
            <label class="switch">
                <input type="checkbox" id="balance-input">
                <span class="slider round"></span>
            </label>
        </div>
        <div class="bar-item">
            <p class="text-hint">Duplicates</p>
            <label class="switch">
                <input type="checkbox" id="duplicate-input">
                <span class="slider round"></span>
            </label>
        </div>
        <div class="line"></div>
        <div class="bar-item">
            <p class="text-hint">Animation</p>
            <label class="switch">
                <input type="checkbox" id="animation-input">
                <span class="slider round"></span>
            </label>
        </div>

        <div class="bar-item">
            <h5>Scale:</h5>
            <input type="range" id="scale-input" min=0.5 max=1.0
step="0.01">
        </div>
        <div class="line"></div>
        <div class="bar-item">
            <input type="text" class="input" id="new-node-input"
placeholder="Add by value">
            <button onclick="addNode()" class="button" id="new-
node-button">Add node</button>
        </div>
        <div class="bar-item">
            <input type="text" class="input" id="remove-node-
input" placeholder="Remove by value">
            <button onclick="removeNode()" class="button"
id="remove-node-button">Remove node</button>
        </div>
        <div class="bar-item">
            <input type="text" class="input" id="remove-id-node-
input" placeholder="Remove by ID">
```

```
            <button onclick="removeNodeByID()" class="button"
id="remove-id-node-button">Remove node</button>
        </div>
        <div class="line"></div>
        <div class="bar-item">
            <label id="input-file-wrapper">
                <p>Choose file</p>
                <input type="file" id="file-input"
accept=".txt"/>
            </label>
            <button onclick="load()" class="button" id="create-
button">Create</button>
        </div>
        <button onclick="download()" class="bar-item button"
id="download-button">Download</button>
        <div class="line"></div>
        <button onclick="deleteAll()" class="bar-item button
red" id="delete-all-button">Delete all</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

//style.css

```
* {
    margin: 0;
    padding: 0;
    font-family: 'Helvetica Neue', Arial, sans-serif;
    font-smooth: never;
}
body {
    display: flex;
}
.block {
    position: absolute;
    display: flex;
    flex-direction: column;
    /* height: 150px;
    width: 250px; */
    background-color:#f8f5f2;
    border-radius: 25px;
    border: solid #232323 3px;
    transition: .2s;
}
.block-value {
    display: flex;
    justify-content: center;
    align-items: center;
    color: #232323;
    font-weight: bold;
    font-size: 20px;
```

```
    overflow: auto;
    flex-grow: 1;
    padding: 10px 6px;
    word-break: break-all;
}
.block-info {
    border-top: #232323 4px dashed;
    display: block;
    text-align: center;
    color: #232323;
    font-weight: bold;
    font-size: 16px;
    height: 16px;
    padding: 6px;
}
#view {
    flex-grow: 1;
    float: left;
    display: block;
    height: 100vh;
    width: 70%;
    background-color: #fffffe;
    overflow: auto;
    cursor: grab;
}
#content {
    transform: scale(1);
    position: relative;
    display: block;
}
#sidebar::-webkit-scrollbar {
    display: none;
}
#sidebar {
    overflow: auto;
    -ms-overflow-style: none;  /* IE and Edge */
    scrollbar-width: none;  /* Firefox */
    display: block;
    float: right;
    height: 100vh;
    width: fit-content;
    background-color: #f8f5f2;
    padding: 0 5px;
    border-left: #232323 4px dashed;
}
.line {
    margin: 0 auto;
    background-color: #232323;
    height: 3px;
    width: 95%;
    margin: 10px auto;
}
```

```
.bar-item {
    display: flex;
    flex-direction: row;
    align-items: stretch;
    width: 100%;
    height: 60px;
    margin: 0 0 5px;
}
.bar-item.button {
    cursor: pointer;
    display: block;
    padding: 0 10px;
    margin: 10px auto;
    border: none;
    background-color: #35FF69;
    border-radius: 5px;
    font-size: 16px;
    font-weight: bold;
    width: 80%;
    height: 50px;
    box-shadow: 1px 1px 1px #0F0F0F;
}
.input {
    background-color: #f8f5f2;
    display: block;
    flex-grow: 1;
    border: 2px #232323 solid;
    margin: 6px;
    font-size: 25px;
    border-radius: 5px;
    padding: 0 3px;
}
.input:focus {
    outline: none;
}
#input-file-wrapper {
    cursor: pointer;
    flex-grow: 1;
    display: block;
    padding: 0 10px;
    margin: 6px 10px;
    line-height: 48px;
    text-align: center;
    border: none;
    background-color: #35FF69;
    border-radius: 5px;
    font-size: 16px;
    font-weight: bold;
    min-width: 60px;
    box-shadow: 1px 1px 1px #0F0F0F;
    transition: background-color 0.2s;
}
```

```
#input-file-wrapper input[type=file] {
     position: absolute;
     z-index: -1;
     opacity: 0;
     display: block;
     width: 0;
     height: 0;
}
#input-file-wrapper:hover {
     background-color: #F9D949;
    transition: background-color 0.2s;
}
#input-file-wrapper:active {
     box-shadow: none;
    transform: translate(1px, 1px);
}
.input::placeholder {
    font-size: 20px;
    font-weight: bold;
    opacity: 0.5;
    color:#535050;
}
.wrong-input-animation {
    animation: blink 0.75s;
}
@keyframes blink{
    0% {
      background: white;
    }
    25% {
      background: #f45d48;
    }
    50% {
      background: white;
    }
    75% {
      background: #f45d48;
    }
    100% {
      background: white;
    }
  }
.button {
    cursor: pointer;
    display: block;
    padding: 0 10px;
    margin: 6px 10px 6px 0;
    border: none;
    background-color: #1ac2c2;
    border-radius: 5px;
    font-size: 16px;
    font-weight: bold;
```

```
    min-width: 110px;
    box-shadow: 1px 1px 1px #0F0F0F;
    transition: background-color 0.2s;
}
.button#delete-all-button {
    background-color: #f45d48;
    transition: background-color 0.2s;
}
.button#delete-all-button:hover {
    background-color: #F9D949;
    transition: background-color 0.2s;
}
.button:hover {
    background-color: #F9D949;
    transition: background-color 0.2s;
}
.button:active {
    box-shadow: none;
    transform: translate(1px, 1px);
}
.switch {
    position: relative;
    display: inline-block;
    width: 60px;
    height: 34px;
    margin: auto auto auto 15px;
}
.switch input {
    opacity: 0;
    width: 0;
    height: 0;
}

.slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #f45d48;
    -webkit-transition: .4s;
    transition: .4s;
    box-shadow: 0 0 1px #0F0F0F;
}
.slider:before {
    position: absolute;
    content: "";
    height: 26px;
    width: 26px;
    left: 4px;
    bottom: 4px;
```

```
    background-color: #fffffe;
    -webkit-transition: .4s;
    transition: .4s;
}
input:checked + .slider {
    background-color: #1ac2c2;
}
input:focus + .slider {
    box-shadow: 0 0 1px #1ac2c2;
    outline: none;
}
input:checked + .slider:before {
    -webkit-transform: translateX(26px);
    -ms-transform: translateX(26px);
    transform: translateX(26px);
}
.notransition {
    -webkit-transition: none !important;
    -moz-transition: none !important;
    -o-transition: none !important;
    transition: none !important;
    -webkit-transform: none !important;
    -ms-transform: none !important;
    transform: none !important;
}
.slider.round {
    border-radius: 34px;
}
.slider.round:before {
    border-radius: 50%;
}
.text-hint {
    text-align: right;
    margin: auto 0 auto auto;
    font-size: 24px;
    font-weight: bold;
    color: #232323;
}
.marked-yellow {
    background-color: #F9D949;
    transition: .2s;
}
.marked-red {
    background-color: #f45d48;
    transition: .2s;
}
.marked-green {
    background-color: #35FF69;
    transition: .2s;
}
.bar-item > h5 {
    margin: auto 0 auto 20px;
```

```
    font-size: 24px;
    font-weight: bold;
    color: #232323;
}
input[type=range] {
    background-color: #f8f5f2;
    margin: 10px 20px 10px 10px;
    -webkit-appearance: none;
    height: 40px;
    width: 100%;
}
input[type=range]:focus {
    outline: none;
}
input[type=range]::-webkit-slider-runnable-track {
    width: 100%;
    height: 14px;
    cursor: pointer;
    animate: 0.2s;
    box-shadow: 0px 0px 1px #0f0f0f;
    background: #232323;
    border-radius: 14px;
    border: 0px solid #f3c846;
}
input[type=range]::-webkit-slider-thumb {
    box-shadow: 0px 0px 1px #0f0f0f;
    border: 0px solid #f3c846;
    height: 20px;
    width: 40px;
    border-radius: 12px;
    background: #1ac2c2;
    cursor: pointer;
    -webkit-appearance: none;
    margin-top: -3px;
}
input[type=range]:focus::-webkit-slider-runnable-track {
    background: #232323;
}
input[type=range]::-moz-range-track {
    width: 100%;
    height: 14px;
    cursor: pointer;
    animate: 0.2s;
    box-shadow: 0px 0px 1px #0f0f0f;
    background: #232323;
    border-radius: 14px;
    border: 0px solid #f3c846;
}
input[type=range]::-moz-range-thumb {
    box-shadow: 0px 0px 1px #0f0f0f;
    border: 0px solid #f3c846;
    height: 20px;
```

```
    width: 40px;
    border-radius: 12px;
    background: #1ac2c2;
    cursor: pointer;
}
input[type=range]::-ms-track {
    width: 100%;
    height: 14px;
    cursor: pointer;
    animate: 0.2s;
    background: transparent;
    border-color: transparent;
    color: transparent;
}
input[type=range]::-ms-fill-lower {
    background: #232323;
    border: 0px solid #f3c846;
    border-radius: 12px;
    box-shadow: 0px 0px 1px #0f0f0f;
}
input[type=range]::-ms-fill-upper {
    background: #232323;
    border: 0px solid #f3c846;
    border-radius: 12px;
    box-shadow: 0px 0px 1px #0f0f0f;
}
input[type=range]::-ms-thumb {
    margin-top: 1px;
    box-shadow: 0px 0px 1px #0f0f0f;
    border: 0px solid #f3c846;
    height: 20px;
    width: 40px;
    border-radius: 12px;
    background: #1ac2c2;
    cursor: pointer;
}
input[type=range]:focus::-ms-fill-lower {
    background: #232323;
}
input[type=range]:focus::-ms-fill-upper {
    background: #232323;
}
```

ПРИЛОЖЕНИЕ Б

Внешний вид графического материала

На рисунке Б.1 представлена диаграмма классов.

На рисунке Б.2 представлена диаграмма компонентов.

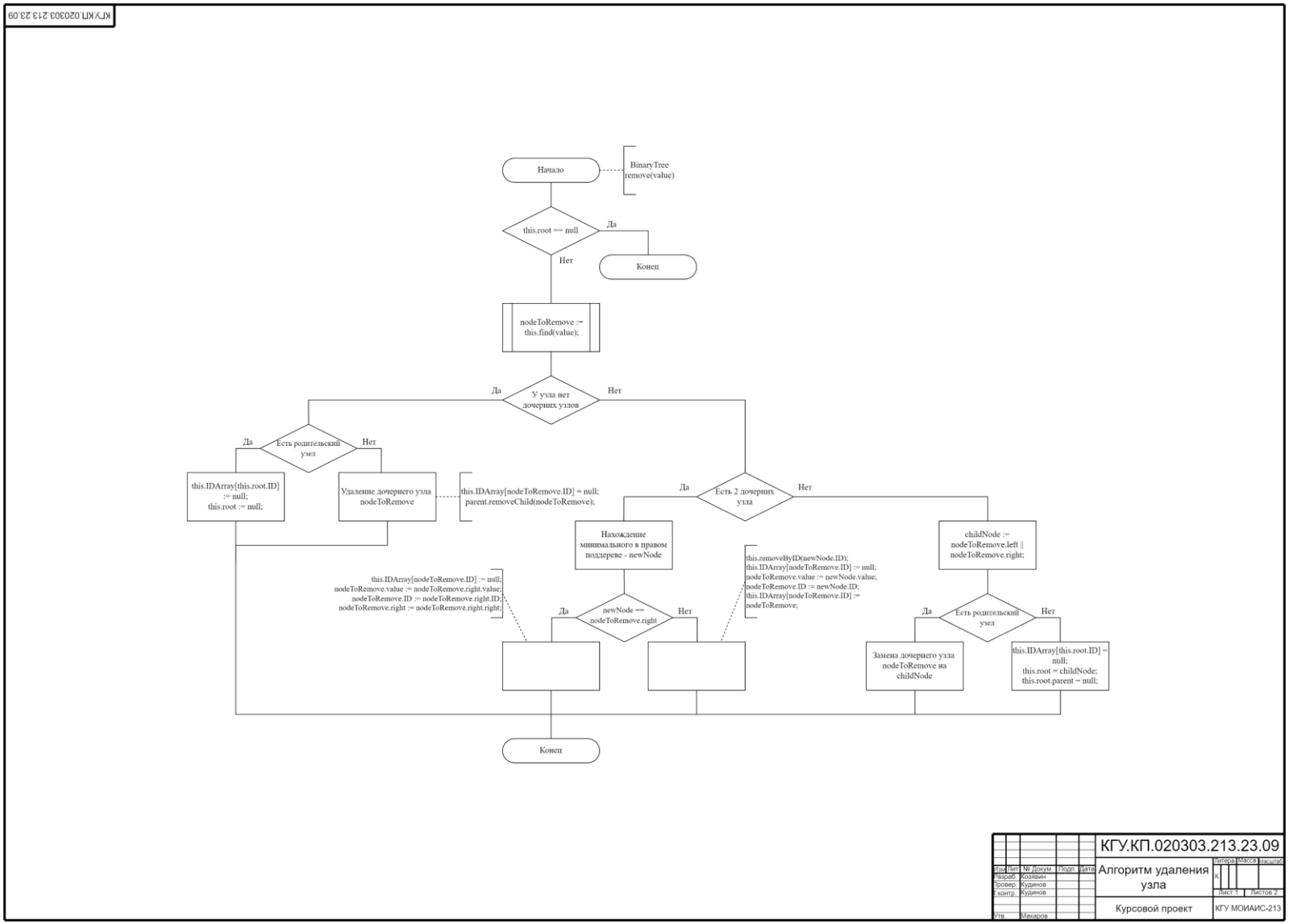

Рисунок Б.1 – Алгоритм удаления узла

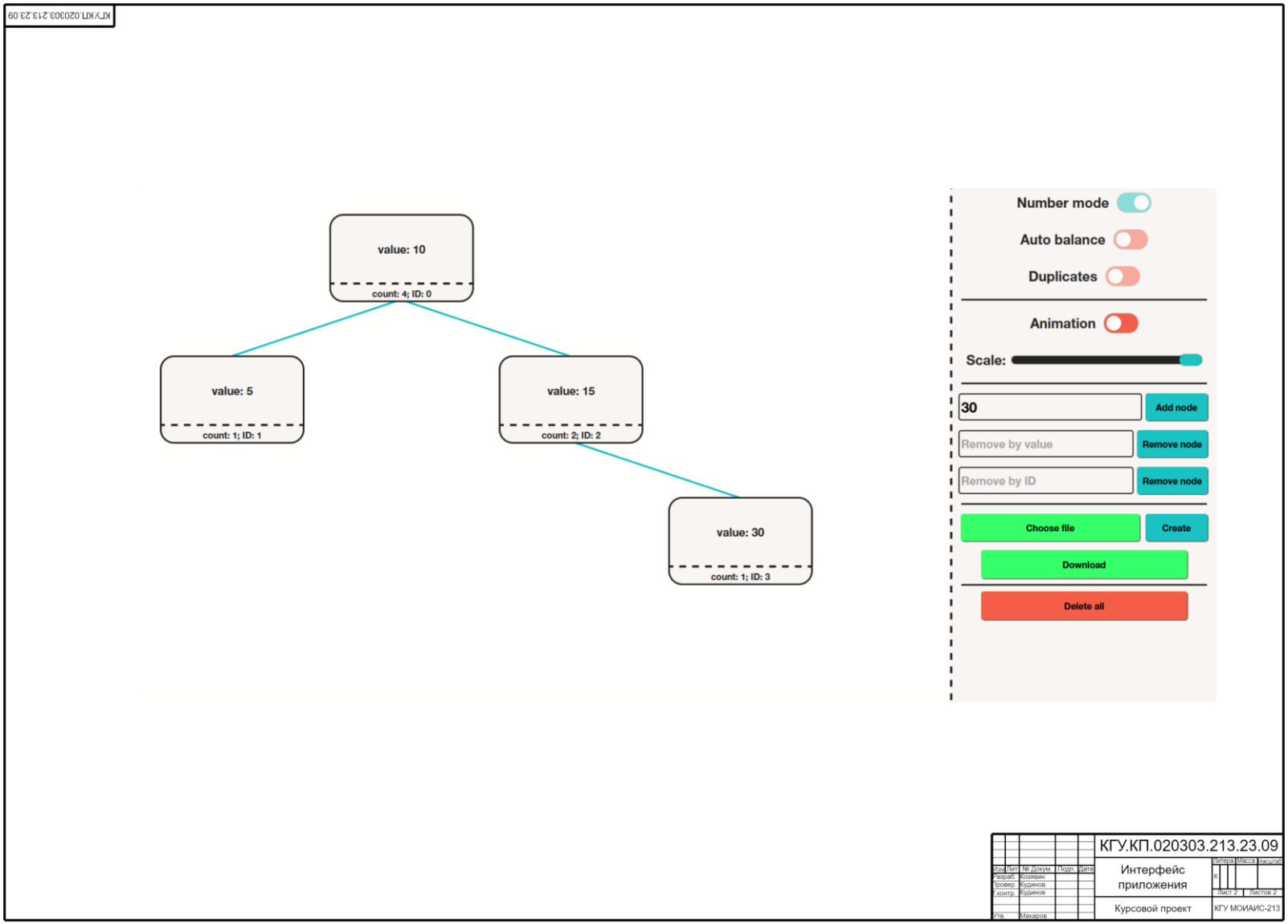

Рисунок Б.2 – Интерфейс приложения